DIP 4th Lecture

September 19, 2023

1 Lecture 4: Digital Image Processing

1.0.1 Line or Path Finding Algorithms using Neighbourhood and Adjacency Processing

1.1 Objectives

- Creating a binary image using numpy
- Show images using matplotlib
- Writing function to find N4, N8 and ND neighbours of a pixel
- Implimenting line or path finding algorithms using N4 and N8 neighbours and Adjaency Processing

---

1.1.1 Helper Function

[ ]:
```
def line_remover(image, line):
    image = image.copy()

    for p in line:
        image[p[0], p[1]] = 1

    return image
```

1.1.2 Importing Libraries

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

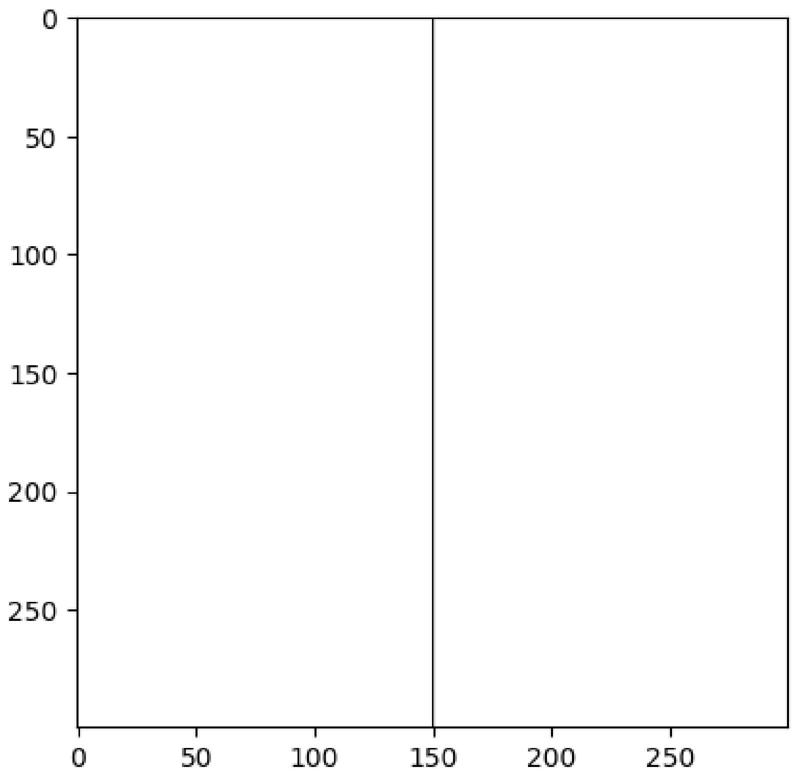
1.1.3 Creating a Binary Image

Create a binary images using numpy array. The image should be of size 300x300 and should have a straight line or path in it. The line or path should be of thickness 1 pixel and should be in the middle of the image.

[ ]:
```
image = np.ones((300, 300))
image[:, 150] = 0

plt.imshow(image, cmap="gray")
```

```
[ ]: <matplotlib.image.AxesImage at 0x1a16ddc0ed0>
```



1.1.4 Finding Neighbours

Write a functions to find N4, N8 and ND neighbours of a pixel. The function should take the pixel location. The function should return the N4, N8 or ND neighbours of the pixel.

N4 Neighbours

```
[ ]: def get4_neighbours(p):
         # print(" ", image[p[0] - 1, p[1]])
         # print(image[p[0], p[1] - 1], image[p[0], p[1]], image[p[0], p[1] - 1])
         # print(" ", image[p[0] + 1, p[1]])

         return [[p[0] - 1, p[1]], [p[0], p[1] - 1], [p[0], p[1] + 1], [p[0] + 1,␣
      ↪p[1]]]


     get4_neighbours([4, 5])
```

```
[ ]: [[3, 5], [4, 4], [4, 6], [5, 5]]
```

N8 Neighbours

```
[ ]: def get8_neighbours(p):
         # print(image[p[0] - 1, p[1] - 1], image[p[0] - 1, p[1]], image[p[0] - 1,␣
      ↪p[1] + 1])
         # print(image[p[0], p[1] - 1], image[p[0], p[1]], image[p[0], p[1] + 1])
         # print(image[p[0] + 1, p[1] - 1], image[p[0] + 1, p[1]], image[p[0] + 1,␣
      ↪p[1] + 1])

         return [[p[0] - 1, p[1] - 1], [p[0] - 1, p[1]], [p[0] - 1, p[1] + 1],␣
      ↪[p[0], p[1] - 1], [p[0], p[1] + 1], [p[0] + 1, p[1] - 1], [p[0] + 1, p[1]],␣
      ↪[p[0] + 1, p[1] + 1]]


     get8_neighbours([4, 5])
```

```
[ ]: [[3, 4], [3, 5], [3, 6], [4, 4], [4, 6], [5, 4], [5, 5], [5, 6]]
```

ND Neighbours

```
[ ]: def getD_neighbours(p):
         # print(image[p[0] - 1, p[1] - 1], " ", image[p[0] - 1, p[1] + 1])
         # print(" ", image[p[0], p[1]], " ")
         # print(image[p[0] + 1, p[1] - 1], " ", image[p[0] + 1, p[1] + 1])

         return [[p[0] - 1, p[1] - 1], [p[0] - 1, p[1] + 1], [p[0] + 1, p[1] - 1],␣
      ↪[p[0] + 1, p[1] + 1]]


     getD_neighbours([4, 5])
```

```
[ ]: [[3, 4], [3, 6], [5, 4], [5, 6]]
```

1.1.5 Line or Path Finding Algorithms

Write a function to find the path or line in the binary image. The function should take the binary image as input and return the coordinates of path or line in the image.

Straight Line or Path

```
[ ]: def line_finder_using_N4(image):
         V = 0
         line = []

         for M in range(0, image.shape[0] - 1):
             for N in range(0, image.shape[1] - 1):
                 if image[M, N] == V:
                     N4 = get4_neighbours([M, N])
                     for nbr in N4:
                         if (nbr[0] > 0 and nbr[1] > 0) and image[nbr[0], nbr[1]] ==␣
      ↪V:
```

```
                        line.append(nbr)
                    break
    return line
```

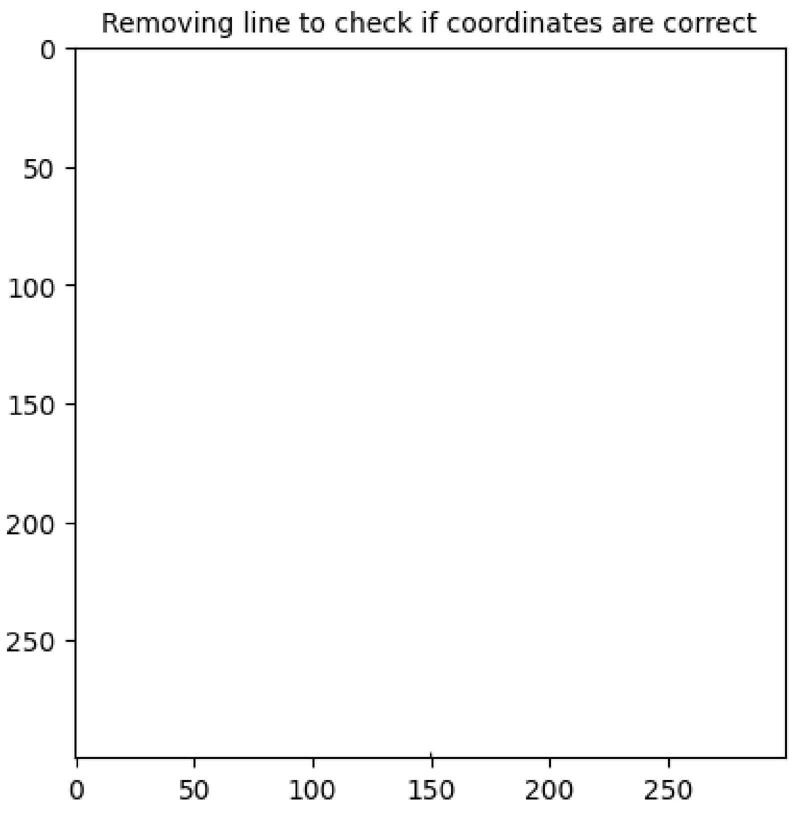
Test the Algorithm As a test case, use the binary image created above and find the path or line in it. To test the algorithm, plot the binary image and remove the path or line found in the image to check, usig helper function line_remover.

[ ]:
```
print("Cordinates of line are:", line_finder_using_N4(image))

plt.title("Removing line to check if coordinates are correct", fontsize=10)
plt.imshow(line_remover(image, line_finder_using_N4(image)), cmap="gray")
```

```
Cordinates of line are: [[1, 150], [2, 150], [3, 150], [4, 150], [5, 150], [6,
150], [7, 150], [8, 150], [9, 150], [10, 150], [11, 150], [12, 150], [13, 150],
[14, 150], [15, 150], [16, 150], [17, 150], [18, 150], [19, 150], [20, 150],
[21, 150], [22, 150], [23, 150], [24, 150], [25, 150], [26, 150], [27, 150],
[28, 150], [29, 150], [30, 150], [31, 150], [32, 150], [33, 150], [34, 150],
[35, 150], [36, 150], [37, 150], [38, 150], [39, 150], [40, 150], [41, 150],
[42, 150], [43, 150], [44, 150], [45, 150], [46, 150], [47, 150], [48, 150],
[49, 150], [50, 150], [51, 150], [52, 150], [53, 150], [54, 150], [55, 150],
[56, 150], [57, 150], [58, 150], [59, 150], [60, 150], [61, 150], [62, 150],
[63, 150], [64, 150], [65, 150], [66, 150], [67, 150], [68, 150], [69, 150],
[70, 150], [71, 150], [72, 150], [73, 150], [74, 150], [75, 150], [76, 150],
[77, 150], [78, 150], [79, 150], [80, 150], [81, 150], [82, 150], [83, 150],
[84, 150], [85, 150], [86, 150], [87, 150], [88, 150], [89, 150], [90, 150],
[91, 150], [92, 150], [93, 150], [94, 150], [95, 150], [96, 150], [97, 150],
[98, 150], [99, 150], [100, 150], [101, 150], [102, 150], [103, 150], [104,
150], [105, 150], [106, 150], [107, 150], [108, 150], [109, 150], [110, 150],
[111, 150], [112, 150], [113, 150], [114, 150], [115, 150], [116, 150], [117,
150], [118, 150], [119, 150], [120, 150], [121, 150], [122, 150], [123, 150],
[124, 150], [125, 150], [126, 150], [127, 150], [128, 150], [129, 150], [130,
150], [131, 150], [132, 150], [133, 150], [134, 150], [135, 150], [136, 150],
[137, 150], [138, 150], [139, 150], [140, 150], [141, 150], [142, 150], [143,
150], [144, 150], [145, 150], [146, 150], [147, 150], [148, 150], [149, 150],
[150, 150], [151, 150], [152, 150], [153, 150], [154, 150], [155, 150], [156,
150], [157, 150], [158, 150], [159, 150], [160, 150], [161, 150], [162, 150],
[163, 150], [164, 150], [165, 150], [166, 150], [167, 150], [168, 150], [169,
150], [170, 150], [171, 150], [172, 150], [173, 150], [174, 150], [175, 150],
[176, 150], [177, 150], [178, 150], [179, 150], [180, 150], [181, 150], [182,
150], [183, 150], [184, 150], [185, 150], [186, 150], [187, 150], [188, 150],
[189, 150], [190, 150], [191, 150], [192, 150], [193, 150], [194, 150], [195,
150], [196, 150], [197, 150], [198, 150], [199, 150], [200, 150], [201, 150],
[202, 150], [203, 150], [204, 150], [205, 150], [206, 150], [207, 150], [208,
150], [209, 150], [210, 150], [211, 150], [212, 150], [213, 150], [214, 150],
[215, 150], [216, 150], [217, 150], [218, 150], [219, 150], [220, 150], [221,
150], [222, 150], [223, 150], [224, 150], [225, 150], [226, 150], [227, 150],
[228, 150], [229, 150], [230, 150], [231, 150], [232, 150], [233, 150], [234,
150], [235, 150], [236, 150], [237, 150], [238, 150], [239, 150], [240, 150],
[241, 150], [242, 150], [243, 150], [244, 150], [245, 150], [246, 150], [247,
150], [248, 150], [249, 150], [250, 150], [251, 150], [252, 150], [253, 150],
[254, 150], [255, 150], [256, 150], [257, 150], [258, 150], [259, 150], [260,
150], [261, 150], [262, 150], [263, 150], [264, 150], [265, 150], [266, 150],
[267, 150], [268, 150], [269, 150], [270, 150], [271, 150], [272, 150], [273,
150], [274, 150], [275, 150], [276, 150], [277, 150], [278, 150], [279, 150],
[280, 150], [281, 150], [282, 150], [283, 150], [284, 150], [285, 150], [286,
150], [287, 150], [288, 150], [289, 150], [290, 150], [291, 150], [292, 150],
[293, 150], [294, 150], [295, 150], [296, 150], [297, 150]]
```

[ ]:
```
<matplotlib.image.AxesImage at 0x1a16de38450>
```



1.1.6 Creating a Binary Image

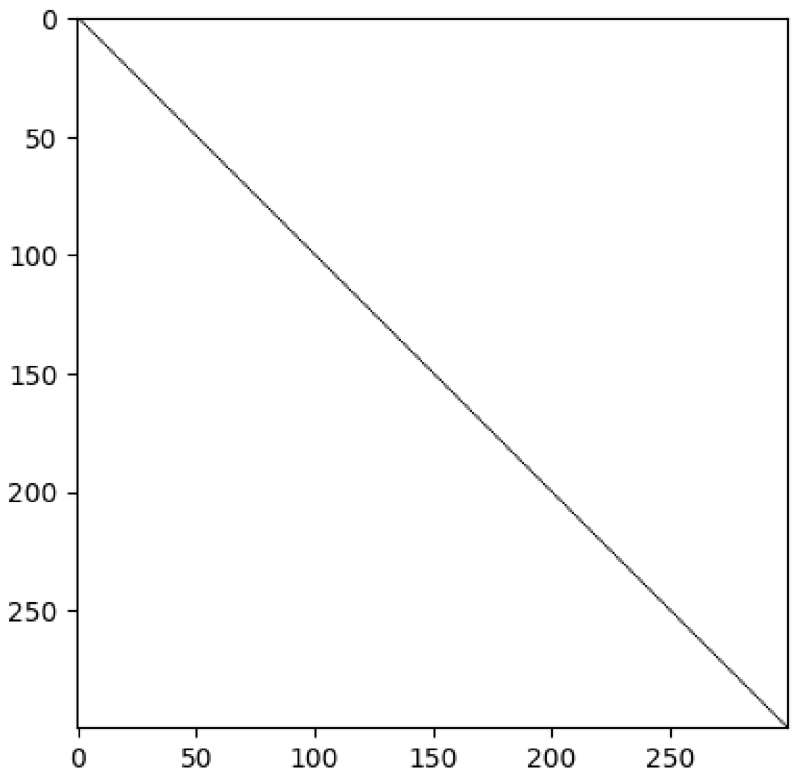
Create a binary images using numpy array. The image should be of size 300x300 and should have a straight line or path in it. The line or path should be of thickness 1 pixel and should be along one of the diagonal in the image.

[ ]:
```
dig_line_image = np.ones((300, 300))

for i in range(0, 300):
    dig_line_image[i, i] = 0

plt.imshow(dig_line_image, cmap="gray")
```

```
[ ]: <matplotlib.image.AxesImage at 0x1a16de90ed0>
```

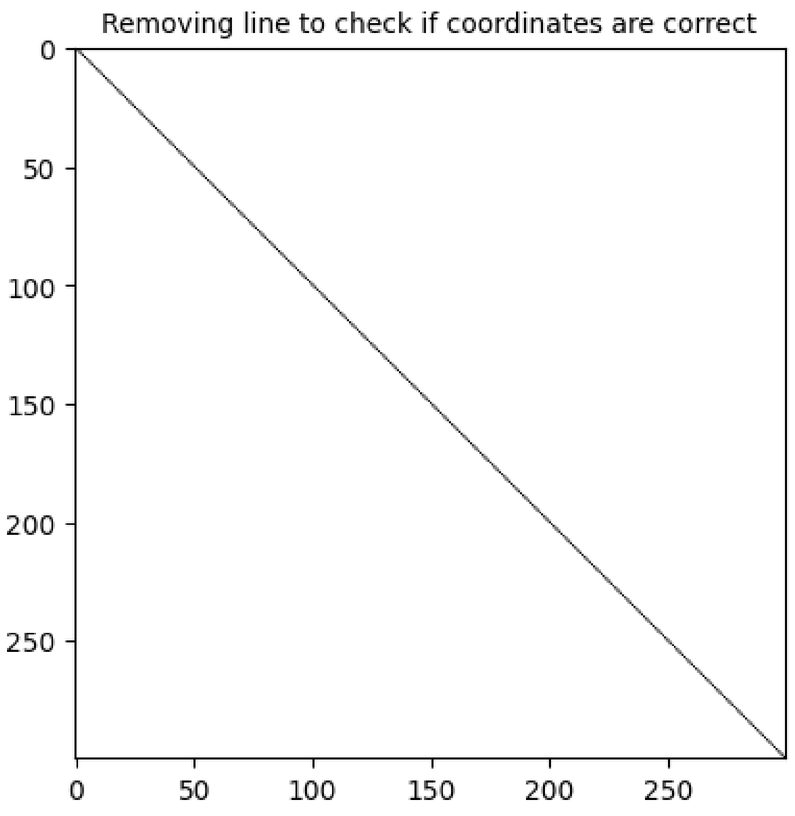


Check As you see line_finder_using_N4 does not work for diagonal lines. This is because the diagonal line does not have any N4 neighbours. To find the diagonal line, we need to use N8 neighbours.

```
[ ]: print("Cordinates of line are:", line_finder_using_N4(dig_line_image))

plt.title("Removing line to check if coordinates are correct", fontsize=10)
plt.imshow(line_remover(dig_line_image, line_finder_using_N4(dig_line_image)),␣
  ↪cmap="gray")
print("It does not work for diagonal line")
```

```
Cordinates of line are: []
It does not work for diagonal line
```



Diagonal Line or Path

```
[ ]: def line_finder_using_N8(image):
         V = 0
         line = []

         for M in range(0, image.shape[0] - 1):
             for N in range(0, image.shape[1] - 1):
                 if image[M, N] == V:
                     N8 = get8_neighbours([M, N])
                     for nbr in N8:
                         if (nbr[0] > 0 and nbr[1] > 0) and image[nbr[0], nbr[1]] ==␣
      ↪V:
                             line.append(nbr)
                         break
         return line
```

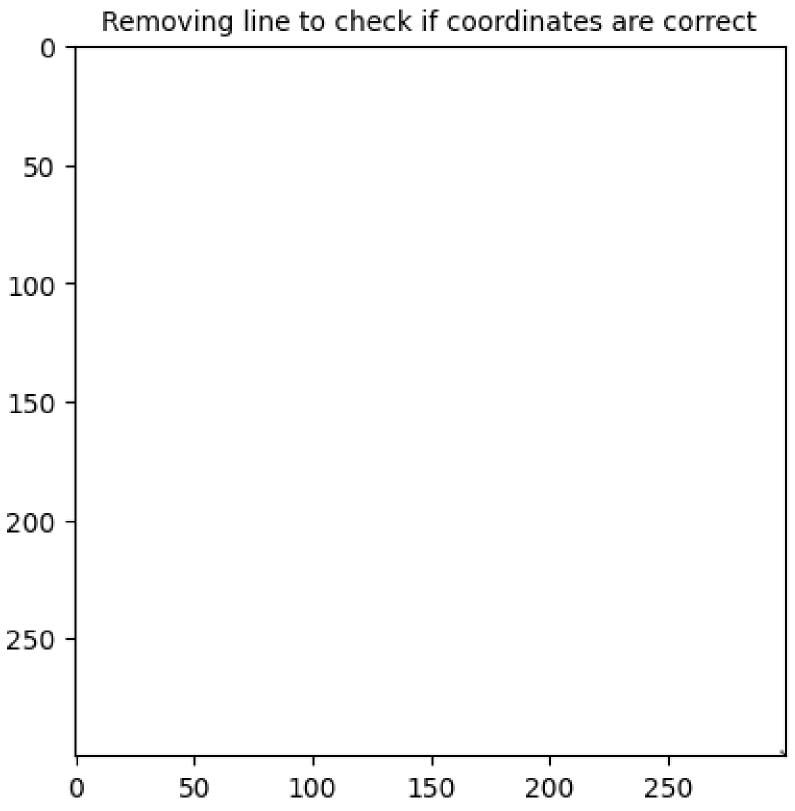
Test the Algorithm As a test case, use the binary image created above and find the path or line in it. To test the algorithm, plot the binary image and remove the path or line found in the image to check, usig helper function line_remover.

```
[ ]: print(line_finder_using_N8(dig_line_image))

     plt.title("Removing line to check if coordinates are correct", fontsize=10)
     plt.imshow(line_remover(dig_line_image, line_finder_using_N8(dig_line_image)),␣
      ↪cmap="gray")
```

```
[[1, 1], [2, 2], [3, 3], [4, 4], [5, 5], [6, 6], [7, 7], [8, 8], [9, 9], [10,
10], [11, 11], [12, 12], [13, 13], [14, 14], [15, 15], [16, 16], [17, 17], [18,
18], [19, 19], [20, 20], [21, 21], [22, 22], [23, 23], [24, 24], [25, 25], [26,
26], [27, 27], [28, 28], [29, 29], [30, 30], [31, 31], [32, 32], [33, 33], [34,
34], [35, 35], [36, 36], [37, 37], [38, 38], [39, 39], [40, 40], [41, 41], [42,
42], [43, 43], [44, 44], [45, 45], [46, 46], [47, 47], [48, 48], [49, 49], [50,
50], [51, 51], [52, 52], [53, 53], [54, 54], [55, 55], [56, 56], [57, 57], [58,
58], [59, 59], [60, 60], [61, 61], [62, 62], [63, 63], [64, 64], [65, 65], [66,
66], [67, 67], [68, 68], [69, 69], [70, 70], [71, 71], [72, 72], [73, 73], [74,
74], [75, 75], [76, 76], [77, 77], [78, 78], [79, 79], [80, 80], [81, 81], [82,
82], [83, 83], [84, 84], [85, 85], [86, 86], [87, 87], [88, 88], [89, 89], [90,
90], [91, 91], [92, 92], [93, 93], [94, 94], [95, 95], [96, 96], [97, 97], [98,
98], [99, 99], [100, 100], [101, 101], [102, 102], [103, 103], [104, 104], [105,
105], [106, 106], [107, 107], [108, 108], [109, 109], [110, 110], [111, 111],
[112, 112], [113, 113], [114, 114], [115, 115], [116, 116], [117, 117], [118,
118], [119, 119], [120, 120], [121, 121], [122, 122], [123, 123], [124, 124],
[125, 125], [126, 126], [127, 127], [128, 128], [129, 129], [130, 130], [131,
131], [132, 132], [133, 133], [134, 134], [135, 135], [136, 136], [137, 137],
[138, 138], [139, 139], [140, 140], [141, 141], [142, 142], [143, 143], [144,
144], [145, 145], [146, 146], [147, 147], [148, 148], [149, 149], [150, 150],
[151, 151], [152, 152], [153, 153], [154, 154], [155, 155], [156, 156], [157,
157], [158, 158], [159, 159], [160, 160], [161, 161], [162, 162], [163, 163],
[164, 164], [165, 165], [166, 166], [167, 167], [168, 168], [169, 169], [170,
170], [171, 171], [172, 172], [173, 173], [174, 174], [175, 175], [176, 176],
[177, 177], [178, 178], [179, 179], [180, 180], [181, 181], [182, 182], [183,
183], [184, 184], [185, 185], [186, 186], [187, 187], [188, 188], [189, 189],
[190, 190], [191, 191], [192, 192], [193, 193], [194, 194], [195, 195], [196,
196], [197, 197], [198, 198], [199, 199], [200, 200], [201, 201], [202, 202],
[203, 203], [204, 204], [205, 205], [206, 206], [207, 207], [208, 208], [209,
209], [210, 210], [211, 211], [212, 212], [213, 213], [214, 214], [215, 215],
[216, 216], [217, 217], [218, 218], [219, 219], [220, 220], [221, 221], [222,
222], [223, 223], [224, 224], [225, 225], [226, 226], [227, 227], [228, 228],
[229, 229], [230, 230], [231, 231], [232, 232], [233, 233], [234, 234], [235,
235], [236, 236], [237, 237], [238, 238], [239, 239], [240, 240], [241, 241],
[242, 242], [243, 243], [244, 244], [245, 245], [246, 246], [247, 247], [248,
248], [249, 249], [250, 250], [251, 251], [252, 252], [253, 253], [254, 254],
[255, 255], [256, 256], [257, 257], [258, 258], [259, 259], [260, 260], [261,
261], [262, 262], [263, 263], [264, 264], [265, 265], [266, 266], [267, 267],
[268, 268], [269, 269], [270, 270], [271, 271], [272, 272], [273, 273], [274,
274], [275, 275], [276, 276], [277, 277], [278, 278], [279, 279], [280, 280],
[281, 281], [282, 282], [283, 283], [284, 284], [285, 285], [286, 286], [287,
287], [288, 288], [289, 289], [290, 290], [291, 291], [292, 292], [293, 293],
[294, 294], [295, 295], [296, 296], [297, 297]]
```

```
[ ]: <matplotlib.image.AxesImage at 0x1a16cc35290>
```



1.2 Conclusion

In this lecture, we learned how to create a binary image and find the path or line in it using N4, N8 and ND neighbours and Adjaency Processing. We also learned how to test the algorithm using helper function line_remover.